CS6310 – Software Architecture & Design
Assignment #4 [150 points]: Streaming Wars Project – Individual Implementation (v3)
Fall Term 2020 – Instructor: Mark Moss

Submission Deliverables

- This assignment must be completed as an individual, not as part of a group.
- You must submit the following items in Canvas:
  (1) A ZIP file named **2020-08-A4.zip** that includes your source code, executable JAR file, and your docker file. The structure of the JAR file is shown in more detail below.
  (2) An updated copy of the UML Design Class Diagram from the earlier assignment reflecting any problem requirement updates as described below, named **class_diagram.pdf**.
  (3) A UML Sequence Diagram that captures the sequence of actions described below, named **sequence_diagram.pdf**.
- Also, ensure that you've submitted the proper files via your Georgia Tech Github Account. Please see Piazza post @509 if you have questions about this aspect of the submission.

- You must notify us via a private post on Canvas and/or Piazza BEFORE the Due Date if you are encountering difficulty submitting your project. You will not be penalized for situations where Canvas is encountering significant technical problems. However, you must alert us before the due date – not well after the fact. You are responsible for submitting your answers on time in all other cases.
- Please consider that uploading files to Canvas might occasionally take a long time, even in the case of seemingly "relatively small" submissions. Plan accordingly, as submissions outside of the Canvas Availability Date will likely not be accepted. You are permitted to do unlimited submissions, thus we recommend you save, upload and submit often. You should use the same file naming standards for the (optional) "interim submissions" that you do for the final submission.

General Intent

In the first phase of the Streaming Wars Project, you were required to provide design artifacts to describe your approach to structuring the classes, attributes, operations, methods and relationships needed to simulate the problem. Now, you are required to provide a lightweight implementation of the system in Java that reflects your design.

You are required to demonstrate fundamental separation of classes in your source code indicative of a reasonable design. This means that your source code should show some indication of being divided up into classes, files, etc. that match your design. You will lose points if you submit poorly structured code.

And don't panic if you realize that your original design had flaws while working on this assignment – this is common for "agile styles" of development. Make sure that your implementation works per these requirements and specifications and note your earlier design oversights with some brief comments in your code. You're also being asked to provide improved design artifacts based on your "revised and improved understanding" of the problem space, and especially in light of the problem requirement changes included in this document.

We provide some examples of the expected input and output for a few test cases. You must provide the actual Java source code, along with all references to any external packages that you used to develop your system. You must also provide an executable JAR file to support the testing and evaluation of your system. We must also be able to recompile your application from your source code as part of the evaluation process.

Problem Scenario
Your requirements for the Streaming Wars Project are continued here. Any requirements from the earlier assignments are carried over here unless explicitly modified and/or otherwise cancelled. If you feel that there are any conflicts between any of the earlier requirements and the more recent requirements, then please seek clarification immediately. In general, though, the newer requirements will take precedence.

This assignment involves implementing some of the core architecture and functional capabilities for the Streaming Wars Project that you designed in the earlier assignments. Your system must implement the following functionality:

(1) The system must initialize the current month and year as 10 (October) and 2020, respectively.
(2) The system must allow the user to create demographic groups, streaming services, movies, Pay-Per-View events, and studios.
(3) The system must allow streaming services to offer movies and Pay-Per-View events.
(4) The system must allow portions of a demographic group to view a streaming service offering.
(5) The system must allow the user to advance the time period to the next month.
(6) The system must calculate and display the monies spent by the demographic groups, and the monies collected by the streaming services and studios for: (a) the current time period; (b) the previous month; and, (c) the total since the start of the program run.

Your system does not have to produce a graphical display. At this point, we will evaluate your system based on the text-based output and the structure of the source code. You'll have ample opportunity to develop a more advanced graphical user interface during the group project.

Evaluation/Grading Your Submissions
- Your submission will be evaluated based on four main areas:

(1) 20 points for your system's correctness: source code provided, working JAR file, the capability to recompile your code if necessary, etc.
(2) 40 points for correct operation of the system on the selected scenario (test) files
(3) 50 points for reasonably structured source code
(4) 40 points for your updated Class & Sequence diagrams

Don't lose sight of the goal for the course: there are many more potential improvements that you could make to this system, but you really need to think about how you will separate and distribute the complexity of the overall problem across separate classes; and, how the objects instantiated from *those classes then communicate and collaborate to solve the problem. Helping you develop a solid structure for your design is the most important aspect of working on this assignment.*

(1) *20 points for your submission correctness: working JAR file, the capability to recompile your code if necessary, etc.*
Common issues that cause you to lose points in this category:

- Not submitting a working JAR file
- The JAR file doesn't function properly on the VM during testing – for example, JNI errors, etc.
- Gross formatting errors – we've designed the testing harness to be as robust as possible when processing the submissions, but errors caused by including graphical displays of the space region, diagnostic output, and other random messages will also cause you to lose points.
- Formatting is important! Syntax is important! Use the matching characters for the output strings, and don't put extra spaces between elements of the output strings. Strings that do not match the correct output because of formatting (syntax) errors might receive significant penalties.

(2) *40 points for correct operation of the system on the selected scenario (test) files*

(3) *50 points for reasonably structured source code*
The main issue that will cause you to lose points in this category is submitting poorly structured, possibly monolithic source code that doesn't display any indication of separation of responsibilities among classes, objects, etc. There isn't a specific set of objects that you must have, but you do need to display some significant effort to apply object-oriented analysis & design principles.

(4) *40 points for your updated Class & Sequence diagrams*
The main issue that will cause you to lose points in this category is submitting Class & Sequence Diagrams that are inconsistent, and/or that don't reflect the latest changes to the problem requirements. For your Sequence Diagram, you only need to show the interactions required to successfully complete the following commands in your system:

- **`create_event`**
- **`offer_event`**
- **`watch_event`**
- **`display_events`**

You should create enough notional objects to demonstrate the messages and related data being sent between these objects.

Input/Output & Command Formatting Requirements

We will test your program at this point using a relatively simple, text-based Command Line Interface (CLI) approach. The commands that we will use are listed below, and your system is expected to follow the syntax of the commands as listed. Your program should accept input from a basic command (terminal) interface – please don't use any windows, third-party processors, etc.

`[1] create_demo,<short name>,<long name>,<number of accounts>`
This command must create a demographic group with the corresponding short name, long name, and number of accounts. The short name must be unique for all demographic groups. For example:

> create_demo,age_40_50,Viewers between 40 and 50,800
This command must create a demographic group that can be referenced in later commands using the short name **age_40_50**, with the long name as shown above, and with 800 accounts.

[2] create_studio,<short name>,<long name>
This command must create a studio with the corresponding short and long names. The short name must be unique for all studios. For example:

> create_studio,disney,Walt Disney Animation Studios
This command must create a studio that can be referenced in later commands using the short name **disney**, and with the long name as shown above.

[3] create_event,<type>,<name>,<year produced>,<duration>,<studio>, <license fee>
This command must create an event type – **movie** or Pay-Per-View (**ppv**) – with the given name, year of production, duration, and licensing fee. The combination of the name and year produced must be unique for all movies. Also, the movie must have been produced by a valid studio. If the studio being referenced is invalid (e.g., hasn't been created yet), then this command should be considered invalid, and the movie should not be created. Also, we will not be concerned with genres at this point. For example:

> create_event,movie,Mulan,1998,88,disney,1000
This command must create a movie named **Mulan** that is 88 minutes long and was produced in the year 1998 by the **disney** studio. Also, any streaming service that wants to offer this movie must pay the studio $1,000 USD per month as the licensing fee. Similarly,

> create_event,ppv,30 for 30: Monaco,2020,106,espn,3300
This command must create a PPV event named **30 for 30: Monaco** that is 106 minutes long and was produced in the year 2020 by **espn** studio. Also, any streaming service that wants to offer this PPV event must pay the studio $3,300 USD as the licensing fee. Of course, this is an example – if there is no valid studio for the short name **espn**, then this PPV event should not be created.

[4] create_stream,<short name>,<long name>,<subscription price>
This command must create a streaming service with the given short name, long name and monthly subscription price. The short name must be unique for all streaming services. For example:

> create_stream,apv,Amazon Prime Video,12
This command must create a streaming service that can be referenced in later commands using the short name **apv,** with a long name as shown above and a subscription price of $12 USD per month.

[5] offer_movie,<streaming service>,<movie name>,<year produced>
This command must create a listing for the designated streaming service that allows prospective viewers (i.e., portions of demographic groups) to access and watch the movie. If either the streaming service or movie name reference is invalid, then this command should be considered invalid, and the listing should not be created. For example:

> offer_movie,apv,Mulan,1998
This command must create a listing that allows viewers to subscribe to the **apv** streaming service to access and watch the 1998 version of the movie **Mulan**. Also, from an accounting or billing perspective, this means that the **disney** studio, which produced the movie, has earned $1,000 USD in licensing fees for that month for this offering.

[6] offer_ppv,<streaming service>,<pay-per-view name>, <year produced>,<viewing price>
This command must create a listing for the designated streaming service that allows prospective viewers (i.e., portions of demographic groups) to access and watch the Pay-Per-View event for the designated price. If either the streaming service or pay-per-view name reference is invalid, then this command should be considered invalid, and the listing should not be created. For example:

> offer_ppv,net,30 for 30: Monaco,2020,57
This command must create a listing that allows viewers to access and watch the Pay-Per-View event **30 for 30: Monaco** via the Netflix (**net**) streaming service for the cost of $57 USD. Also, from an accounting or billing perspective, this means that the **espn** studio, which produced the Pay-Per-View event, has earned $3,300 USD in licensing fees for that month for this offering.

[7] watch_event,<demographic group>,<percentage>, <streaming service>,<event name>,<year produced>
This command must create a transaction such that some percentage of the accounts in the designated demographic group have decided to access and watch the movie or Pay-Per-View event being offered on the streaming service. As with most commands, if the references to the demographic, streaming service or event are invalid, then this command should also be considered invalid. Similarly, if the streaming service is not offering the designated event, then the command should be considered invalid. For example:

> watch_event,age_40_50,30,apv,Mulan,1998
This command must create a transaction such that 30% of the accounts in the **age_40_50** demographic have accessed and watched the **Mulan** movie via the **apv** streaming service. From an accounting or billing perspective, this means that 0.30 * 800 = 240 accounts have had to subscribe to the **apv** service such that the **apv** service has earned 240 * $12 USD = $2,880 USD in subscription fees, assuming that this is the first movie for those subscribers during the current month.

We are going to make an assumption to make the calculations at this phase of the project a bit simpler. Suppose the **age_40_50** demographic decides to watch more movies from the **apv** streaming service during the current month. Let the next relevant command be:

> watch_event,age_40_50,50,apv,Beauty and the Beast,1991
We will always make the assumption that the percentage of accounts viewing the movie includes as many previous subscribers as possible. In this case, since 240 accounts (30% of the group) already watched a movie, then we'll assume that they are among the 50% of the movie watcher in this current command. Consequently, we will only need to charge more subscription fees for the "new 20%" of

the accounts (50% * 800 – 240 = 160) that are watching this second movie. Just to complete the example, let the next relevant command be:

```
> watch_event,age_40_50,40,apv,The Little Mermaid,1989
```

Once again, we will always make the assumption that the percentage of accounts viewing the movie includes as many previous subscribers as possible. In this case, since 50% of the group has already paid subscriber fees, we will safely assume that the 40% who are watching the third movie have all already subscribed to the **apv** streaming service, and so there are no new subscription fees to be collected.

[8] next_month

This command advances the calendar to the next calendar month. If the current time period is 8/2020, then the next time period will be 9/2020. If the current time period is 12/2020, then the next time period will be 1/2021. The system will start with default time period of October 2020 (10/2020). This change in time period has implications for state of the system.

- All **creation** actions that have been executed successfully – to include creating demographic groups, streaming services, studios, movies and Pay-Per-View Events – must automatically persist from month to month. They do not need to be recreated each month.
- The movies and Pay-Per-View events that were offered via the **offer_movie** and **offer_ppv** commands must be cancelled automatically at the end of the month. They can and should be recreated as needed by the streaming services.
- All **watch_event** commands are "singular actions", and only pertain to viewers watching a designated event during the month where the command was executed. There is no obligation for viewers to watch events from the same streaming service – or any streaming service for that matter – during the following month. This also means that subscriptions are charged month by month: if viewers do not watch any movies from a particular streaming service during the month, then they must not be charged the subscription fee for that streaming service for that month.

[9] display_demo,<short name>

This command displays information about the demographic group that is referenced by the provided short name. The command must display the information in this format:

```
demo,<short name>,<long name>
size,<number of accounts>
current_period,<spending so far in the current month>
previous_period,<spending in the previous month>
total,<spending for all previous months except the current month>
```

Consider the following example:

```
> display_demo,age_40_50
demo,age_40_50,Viewers between 40 and 50
size,800
current_period,2880
previous_period,0
total,0
```

[10] display_stream,<short name>

This command displays information about the streaming service that is referenced by the provided short name. The command must display the information in this format:

```
stream,<short name>,<long name>
subscription,<subscription fee>
current_period,<subscription fees in the current month>
previous_period,<subscription fees in the previous month>
total,<subscription fees for all previous months except the current month>
licensing,<costs for all months since the program started>
```

Consider the following example:

```
> display_stream,apv
stream,apv,Amazon Prime Video
subscription,12
current_period,2880
previous_period,0
total,0
licensing,4000
```

[11] display_studio,<short name>

This command displays information about the studio or publishing group that is referenced by the provided short name. The command must display the information in this format:

```
studio,<short name>,<long name>
current_period,<licensing fees so far in the current month>
previous_period,<licensing fees in the previous month>
total,<licensing fees for all previous months except the current month>
```

Consider the following example:

```
> display_studio,disney
studio,disney,Walt Disney Animation Studios
current_period,4000
previous_period,0
total,0
> display_studio,espn
studio,espn,Entertainment Sports Network Studios
current_period,3300
previous_period,0
total,0
```

[12] display_events

This command displays all of the movies and Pay-Per-View events that have been produced by the studios. The requests must be listed in the order that they were entered. The command must display the information in this format:

```
<type>,<name>,<year produced>,<duration>,<studio>,<license fee>
...
<type>,<name>,<year produced>,<duration>,<studio>,<license fee>
```

Consider the following example:

```
> display_events
movie,Mulan,1998,88,disney,1000
ppv,30 for 30: Monaco,2020,106,espn,3300
```

```
movie,The Little Mermaid,1989,83,disney,2000
movie,Beauty and the Beast,1991,84,disney,1000
ppv,MMA Championship,2020,121,espn,8800
```

[13] display_offers
This command displays all of the license requests made from the streaming services to the studios for various movies and Pay-Per-View events. The requests must be listed in the order that they were entered. The command must display the information in this format:

```
<stream>,<type>,<short name>,<year>,<subscription price (only) if ppv>
...
<stream>,<type>,<short name>,<year>,<subscription price (only) if ppv>
```

Consider the following example:

```
> display_offers
apv,movie,Mulan,1998
apv,movie,The Little Mermaid,1989
apv,movie,Beauty and the Beast,1991
net,ppv,30 for 30: Monaco,2020,57
```

[14] display_time
This command displays the month and year for the current time period in this format:

```
time,<month>,<year>
```

Consider the following example:

```
> display_time
time,1,2021
```

[15] stop
This command simply causes the (otherwise infinite) interactive loop to halt.

Additional commands to create, update, delete and display elements of the system will be added during later phases of the project.

Sample Code

We've included some sample Java source "shell code" that you may use to better understand the problem space and some of the details in the calculations. The code, though not necessarily well-designed, does perform the required commands correctly.

You are also welcomed to use portions of the code in your own solution, but please note that this code is deliberately not "well designed" by object-oriented standards. If you simply submit this code back to us as your solution, you will most likely forfeit all of the points for having a well-designed solution, and it's also likely you'll lose more points because the code won't match your Class & Sequence Diagrams. Also, use of the provided shell code is 100% optional: you are NOT required/obligated to use this code.

If you would like to take a closer look at the shell code, then the best way might be to create a new Java project with blank Main and TestCodeReader files, and then import the provided content.

Similarly, you can compile and experiment with the code, and create a new JAR file as well. We've also provided an initial copy of the JAR file that can be executed in two ways:

[1] Interactive (Type in the commands yourself one by one):

```
$ java -jar streaming_wars.jar
create_demo,age_40_50,Viewers between 40 and 50,800
> create_demo,age_40_50,Viewers between 40 and 50,800
display_demo,age_40_50
> display_demo,age_40_50
demo,age_40_50,Viewers between 40 and 50
size,800
current_period,0
previous_period,0
total,0
...
display_stream,net
> display_stream,net
stream,net,Netflix
subscription,14
current_period,0
previous_period,0
total,9120
licensing,3300
stop
> stop
$_
```

[2] Scripted (Enter the commands into a text file as demonstrated with the provided example):

```
$ java -jar streaming_wars.jar < streaming_wars_test.txt
> create_demo,age_40_50,Viewers between 40 and 50,800
> display_demo,age_40_50
demo,age_40_50,Viewers between 40 and 50
size,800
current_period,0
previous_period,0
total,0
…
> display_stream,net
stream,net,Netflix
subscription,14
current_period,0
previous_period,0
total,9120
licensing,3300
> stop
$_
```

Also, the test cases that we will use will be fundamentally consistent, and will not include any errors like duplicate entries, or references to non-existent entities/objects.

Submission Details

- You must submit a single ZIP file with the following structure:
  2020-08-A4
  - src
    - bunch of java files
  - submission
    - streaming_wars.jar
  - bin
    - streaming_wars_initial.jar
  - scripts
    - test.sh
    - copy.sh
  - test_scenarios
    - command_x.txt
  - readme.md
  - dockerfile

- You must use the copy.sh before zipping the folder which will create a streaming_wars.jar under the 2020-08-A4/submission folder.

- Make sure that you have run the copy.sh script to create your submission jar zip for the whole top level 2020-08-A4 directory.

- Your system must complete each test case in less than three (3) seconds (real time) – otherwise, your result will be considered to be a failure for that scenario. The scale of these scenarios shouldn't require more computing resources than this – in fact, many systems from the previous term completed all of the test cases (e.g., 20+ cases) in less than three (3) seconds total. Let us know if you feel that you are unable to develop your system to meet this requirement.

- Your program should display the output directly to "standard output" on the command line or terminal – not to a file, special console, etc. And you are welcome to use diagnostic output when testing and troubleshooting your program; however, you must disable or otherwise remove any "excess output" before you submit your program. Any excess (i.e. non-required) output will disrupt the automated evaluation of your system, and requests to re-evaluate your system on this basis will be penalized heavily.

- On a related note, it's highly suggested that you test your finished system on a separate machine if possible, away from the original development environment. Unfortunately, we do receive otherwise correct solutions that fail during our tests because of certain common errors:
  - forgetting to include one or more external libraries, etc.
  - having your program read files from a specific, hardcoded (and non-existent) folder
  - having your program read test files embedded files in its own JAR package

- Many modern Integrated Development Environments (IDEs) such as Eclipse, IntelliJ, Android Studio, etc. offer very straightforward features that will allow you to create a runnable JAR file fairly easily. Also, please be aware that we might recompile your source code to verify the functionality & evaluation of your solution/JAR file compared to your submitted source code.

Closing Comments & Suggestions

This is the information that has been provided by the customer so far. We (the OMSCS 6310 Team) will likely conduct an Office Hours where you will be permitted to ask us questions in order to clarify the client's intent, etc. We will answer some of the questions, but we will not necessarily answer all of them. Also, though this current version of the system if "the core" of the system going forward, our clients will very likely add, update, and possibly remove some of the requirements over the span of the course. One of your main tasks will be to ensure that your architectural documents and related artifacts remain consistent with the problem requirements – and with your system implementations – over time.

Quick Reminder on Collaborating with Others

Please use Piazza for your questions and/or comments, and post publicly whenever it is appropriate. If your questions or comments contain information that specifically provides an answer for some part of the assignment, then please make your post private first, and we (the OMSCS 6310 Team) will review it and decide if it is suitable to be shared with the larger class. Best of luck on to you this assignment, and please contact us if you have questions or concerns.